(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 2 307 968 B1**

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
07.01.2015 Bulletin 2015/02

(21) Application number: **08874834.8**

(22) Date of filing: **25.06.2008**

(51) Int Cl.:
G06F 13/10 *(2006.01)* ***G06F 9/445*** *(2006.01)*

(86) International application number:
PCT/US2008/007898

(87) International publication number:
WO 2009/157896 (30.12.2009 Gazette 2009/53)

(54) DISTRIBUTED CONTROL OF DEVICES USING DISCRETE DEVICE INTERFACES OVER SINGLE SHARED INPUT/OUTPUT

VERTEILTE GERÄTESTEUERUNG UND DISKRETE GERÄTESCHNITTSTELLE ÜBER GETEILTE EIN/AUSGABE DAFÜR

CONTRÔLE RÉPARTI DE DISPOSITIFS À L'AIDE D'INTERFACES DE DISPOSITIF DISCRÈTES SUR UNE ENTRÉE/SORTIE PARTAGÉE UNIQUE

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MT NL NO PL PT RO SE SI SK TR

(43) Date of publication of application:
13.04.2011 Bulletin 2011/15

(73) Proprietor: **GVBB Holdings S.A.R.L**
2086 Luxembourg (LU)

(72) Inventor: **WILSON, Gerald, Philip**
Camptonville, California 95922 (US)

(74) Representative: **Hill, Justin John et al**
Olswang LLP
90 High Holborn
London
WC1V 6XX (GB)

(56) References cited:
EP-A- 1 513 057
US-A1- 2004 216 143
US-A1- 2006 248 526
US-B1- 7 383 386

Note: Within nine months of the publication of the mention of the grant of the European patent in the European Patent Bulletin, any person may give notice to the European Patent Office of opposition to that patent, in accordance with the Implementing Regulations. Notice of opposition shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

EP 2 307 968 B1

Description

[0001] The present disclosure relates generally to systems and methods for accessing and controlling devices in a computing network using virtual device interfaces and, in particular, systems and methods for independently accessing and controlling discrete functionalities of a controlled device through separate virtual device interfaces utilizing a single shared deterministic I/O connection.

Background

[0002] Technological innovations and developments in distributed computing systems and applications have led to a significant increase in the amount and variety of network computing devices and peripheral (hardware) devices that can be interconnected and remotely accessible over communications networks. In general, the functions of network devices can be accessed and controlled by client applications through some type of device interface. For example, a common approach for enabling application access and control of a multi-function device is by use of "virtual device interfaces" in which various functions of the device are exposed through discrete interfaces, where each virtual device interface is an abstraction of actual device functionality. A fundamental goal for a virtual device interface framework is to allow for generic device control code within the application which lends itself to specialization, (the grouping and segregation of specific functional domains), and to the ability of substituting "plug-in" simulators in lieu of the actual device interface. Indeed, it is preferable for the device interface code to maintain a strict hierarchal base that segregates discrete functional (virtual) device types and, thus, the APIs, which leads to more "code reuse," and standardized APIs.

[0003] For example, European Patent Application Publication Number EP1513057discloses a software-aided storage device emulation in a physical storage device, and United States Patent Application Publication Number US2006/0248526 to Rostampour discloses providing virtual device access via firmware.

[0004] As multi-function devices become more complex, however, the ability to abstract device functionality becomes more problematic at the level of both device interface development and application development, especially when deterministic behavior is required. For example, challenges that are faced at the device interface level include: 1. Mode selection - Operations for each of the discrete functionalities often require switching modes of operation with resulting increased complexity; 2. Priority inversion - The operation of a functional block of low importance preempting the operation of another functional block of higher importance; and 3. Increased complexity of device API.

[0005] Conversely, when developing an application that must make use of a device interface which combines several functionalities, the application developer is often required to "understand" the multi-function behavior of the "special device." A common approach to addressing this issue is to create an interface to the interface; one that must be "understood" at the user application level of operation. This direct method is problematic for various reasons. The most obvious being that the client application must include knowledge about a specific device type rather than a set of virtual device types. Ultimately, this knowledge leads to a mass of specialized device-specific code that should not be part of the application domain. In short, the application developer must include the "how" (knowledge of the device control) with the "what" (desired device behavior), which is not desirable. In addition to the application developer becoming responsible for device control issues, this approach inevitably introduces procedures that are tailored to a specific device, and many problems are only exposed when the application must be modified to make use of another device, (or set of devices), to accomplish the same task.

[0006] As the desired approach is to develop applications with "virtual device" interfaces, it is up to the device interface developer to provide the independent, yet integrated, functionality of the complex device. However, this often requires independent control channels for I/O, which are sometimes unavailable. For example, most video servers present two completely independent functions: 1) a "transport" (play/record functionality), and 2) a media management functionality. An interface user, (a scheduling/playout application), may want to manage the resources using a "system interface" and also cue and play clips using transport interfaces. That is, from the prospective of the client application, there are two (or more) discrete functionalities provided by separate devices. This interface scheme reduces the complexity of each device interface by restricting the problem domain to that of a single functionality, and thereby presents an opportunity to model the interface, and the client application, in an object orientated manner. However, this approach often presents a problem when independent control channels for I/O are not available for each operating device interface or when the target device limits the communication by its protocol implementation and/or by its physical/logical connections.

[0007] For example, the LOUTH (VDCP) protocol consists of a mixture of system (resource) and control (transport) commands. A VDCP device, therefore, provides several play/record (control) ports and a system port. However, the system port is only accessible via a control port. Thus, the interface developer is faced with the dilemma of having to present itself as both a "VTR" and a system resource manager. This complicates the interface by forcing modes of operation with priority impacts, and also complicates the application making use of the interface by requiring a knowledge of which interface is the system port.

[0008] Moreover, several major problems with the development of multiple device interfaces when they all op-

erate on a single (physical) device communication connection, as each interface competes for a limited amount of communication, (and CPU), bandwidth. Device interfaces often operate in a quiescent state of "ready" with bursts of "high priority" and time critical activity. When more than one of such interfaces is operating simultaneously, there must be a way to arbitrate both, the maximum usage of the limited communications bandwidth, and the precise timing of time-critical messages, often at a "subfield" resolution. These requirements present three major problems for device interface development, namely, 1. avoidance of process/thread priority inversion or contention; 2. precise timing of certain control messages; and 3. handling timing collisions for multiple control messages.

[0009] Priority inversion refers to the problem of a high priority interface executing a low priority task and taking precedence over a lower priority interface executing a high priority task. Dealing with priority inversion issues is difficult even in a real time operating system where process/thread priorities allow adjusting priorities. Indeed, priority adjustment at runtime is a very difficult and seldom successfully used method for handling the problem of multiple interfaces with bursts of "high priority" activity.

Summary

[0010] In general, systems and methods according to present principles are provided for independently accessing and controlling discrete functionalities of a controlled device through separate virtual device interfaces that utilize a single shared deterministic I/O connection under control of a device I/O module. In one aspect, a method comprises providing a plurality of virtual device interfaces for a target device, wherein each virtual device interface enables client control of a discrete function of the device, providing a single shared I/O connection; and controlling concurrent communication between the target device and two or more virtual device interfaces operating to operate on the single shared I/O connection using a device input/output (I/O) interface to access the target device, wherein timed commands may be sent to the target device. The device I/O module enables development of multiple device interfaces that can seamlessly and independently operate on a single (physical) device communication connection, wherein the device shared I/O module implements functions to arbitrate and coordinate I/O activity between the device interfaces and target device.

[0011] An aspect of the invention is a program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps for controlling a target device, the method steps comprising using a plurality of virtual device interfaces for the target device, wherein each virtual device interface enables client control of a discrete function of the target device; using a single shared I/O connection to the target device; and controlling concurrent communication between the target device and two or more virtual device interfaces to operate on the single shared I/O connection using a device input/output (I/O) interface to access the target device, wherein timed commands may be sent to the target device.

[0012] An aspect of the invention is a system, comprising a component that interfaces with a plurality of virtual devices and a target device to concurrently control communications over a single connection between two or more of the virtual devices and the target device; a front end interface that interfaces with the virtual devices and determines message queues for the virtual devices; and a device input/output (I/O) module having a front end interface, an input/output (I/O) port interface that interacts with a target device based on the queued messages and a stable time base.

[0013] These and other exemplary embodiments, aspects, features, and advantages of the present invention will become apparent from the following detailed description of the preferred embodiments, which is to be read in connection with the accompanying drawings.

[0014] The invention is as defined by the attached independent claims 1, 5 and 9.

Brief Description of the Drawings

[0015]

FIG. 1 is a high-level schematic block diagram of a system for controlling and accessing a physical device using a plurality of virtual device interfaces over a shared communications port, according to an exemplary embodiment of the present principles.

FIG. 2 is a schematic block diagram of a device I/O module according to an exemplary embodiment of the present principles.

Detailed Description

[0016] Exemplary systems and methods for accessing and controlling a device in a distributed computing environment through separate virtual device interfaces that can concurrently communicate with the associated device over a single shared I/O port will now be described in further detail with reference to FIGs. 1 and 2. It is to be understood that the exemplary systems and methods discussed herein can be implemented in various forms of hardware, software, firmware, special purpose processors, or a combination thereof. For example, device interfaces and APIs can be implemented in software as an application comprising program instructions that are tangibly embodied on one or more program storage devices (e.g., magnetic floppy disk, RAM, CD ROM, ROM, Flash memory, etc.) and executable by any computing device, machine or platform comprising suitable architecture. It is to be further understood that because some of the system components and method steps depicted

can be implemented in software, the actual connections between the system components (or the process steps) can differ depending upon the manner in which the present invention is programmed.

[0017] Referring now to FIG. 1, a high-level schematic block diagram is depicted of a system (10) for controlling and accessing a physical device using a plurality of virtual device interfaces over a shared communications port, according to an exemplary embodiment of the present principles. In general, FIG. 1 illustrates a controlled device (11), a device I/O module (12), a plurality of device interface modules (13) and an I/O API module (14). The various interfaces (12), (13) and (14) provide a framework under which clients (applications) can independently access and control discrete functionalities presented by the single controlled device (11) through separate virtual device control interfaces (13) that can concurrently send/receive messages to/from the device (11) over a single deterministic shared I/O connection (15) managed by the device I/O module (12).

[0018] In particular, each virtual device interface (13) associated with the device (11) exposes an interface to a set of abstract functions to access and control an independent, discrete function of the target device (11). The device interfaces (13) provide the necessary function commands that abstract the operating protocol (commands, control signals) of the device for access and control of a discrete function. The device interfaces (13) can concurrently communicate (send and receive messages) with the device (11) via the device I/O module (12), wherein each of the device interfaces (13) can simultaneously access (share) and communicate with the device I/O module (12) via the common I/O API (14). In one exemplary embodiment, the device I/O module (12) comprises a server front end that allows multiple clients (device interfaces (13)) to access the device I/O module (12) via the common I/O API (14), wherein the clients can register interest in all or part of resulting I/O activity during a session. The common I/O API module (14) allows a client using a device interface (13) to send timed (deferred) commands to the target device (11) via the device module (12). The device I/O module (12) comprises a back end device I/O interface that provides the interface functionality (socket, serial port, etc) to interface with the device (11) and send/receive messages to/from the target device (11) over a single shared I/O port (15). In this regard, the device I/O module (12) with the common I/O API module (14) operates as a shared I/O interface providing a many-to-one/one-to-many I/O interface.

[0019] The above exemplary embodiment of FIG. 1 is a high-level conceptual system framework for device control that can be implemented in various application environments and distributed computing systems. FIG. 1 further depicts a method for controlling a device according to present principles by exposing a plurality of virtual device interfaces for a device, wherein each virtual device interface enables client control of a discrete function of the device, and controlling concurrent communication between the device and two or more virtual device interfaces operating on a single shared I/O connection using a device I/O interface.

[0020] For instance, in the above mentioned example of video server applications, the device interfaces (13) can be configured to control independent functions such as (i) a "transport" (play/record) functionality and (ii) a media management functionality. An interface user, (a scheduling/playout application), can access and manage resources using a "system interface" and cue and play clips using a transport interface. From the perspective of the user, the virtual interface device scheme makes the single device access appear to be two (or more) separate devices.

[0021] In contrast to conventional schemes as discussed above where the target device limits the communication by a protocol implementation and/or by a physical/logical connection of the device which forces the interface and application developers to address I/O control and connectivity with device specific knowledge regarding the device communication protocol, the present principles provide solutions to these problem by using a shared device I/O module to implement functions for controlling a shared I/O connection between multiple independent interfaces.

[0022] In accordance with present principles, a shared device I/O module enables development of multiple device interfaces that can seamlessly and independently operate on a single (physical) device communication connection, wherein the device shared I/O module implements functions to arbitrate and coordinate I/O activity between the device interfaces and target device to achieve maximum usage of a limited amount of communications bandwidth, and precise timing of (time critical) messages, often at a "sub-field" resolution.

[0023] FIG. 2 is a block diagram that schematically illustrates a framework of a device I/O module according to an exemplary embodiment of the invention. The device I/O module (120) comprises a front end interface (121) and an I/O port interface (122) and a stable time base (123), which provides a framework by which multiple device interfaces can establish one-to-one communication links (128) to the shared device I/O module (120) and concurrently and independently access a controlled device (110) over a single shared connection (115) to the controlled device (110). As explained below, the device I/O module (120) controls communications in a prioritized manner using a control scheme in which the device interfaces are assigned equal (run time) priority and priority assignment is on a message-level assignment and wherein a message timeline and advance collision detection functions are employed to delegate (combine) high priority activities at a constant priority and handle (arrange) the communications in a prioritized manner.

[0024] In particular, the front end interface (121) comprises a client database (124), a plurality of client message queues (125), a timeline (126) (or transmit queue) and receive queue (127). The client DB (124) is used to

store client information for a given user session for which a client engages the controlled device, wherein such information includes user requests, and other session data associated with I/O activity and status for each client. The client DB (121) allows a client to register information and interest in all or a portion of the I/O events.

[0025] The front end interface (121) implements a message queuing process that temporarily stores incoming client messages from each client operating a given device interface into a corresponding one of the client message queues (125). The message queuing process takes messages from the client message queues (125) and places the messages into a timeline (126) (or transmit queue) where the messages are placed in order of time and priority based on a specification for the priority and timing of messages. The stable time base (123) provides a quantum of time with subdivisions as defined by the system (e.g., a video frame (with quarter-frame subdivisions). In this example, once every quarter frame, the content of the timeline (transmit) queue (126) is evaluated and relevant messages transmitted to the controlled device (110) via the shared I/O port interface (122) over the single I/O connection (115). The receive queue (127) collects responses from the shared I/O port interface (122) which are received from the controlled device (110), and copies the received message to individual client receive queues in accordance with the client's specification of interest.

[0026] The I/O API (14) includes (but is not limited to) a client form that allows one or more multiple clients to register their interest in all or some of the received messages, a specification for the priority and timing of messages, a request for received messages, timeline control and status, all under a framework that enables multiple clients to concurrently access, control and otherwise communicate with a controlled device. For instance, with regard to priority, the front end interface (121) assigns equal priority to virtual device interfaces executing tasks, and prioritizes messages from the virtual device interfaces based on the priority (assigned by the virtual device interface) of the message. The front end interface (121) implements a message-level priority scheme wherein messages are prioritized according to the type of message and/or the importance of an associated task. For instance, control messages and data messages are different types of I/O messages where pending control messages can be afforded higher priority to provide important control tasks, while data messages can be afforded lower priority. Moreover, messages can be assigned priority levels based on critical timing factors such as client requested deadlines, etc. By operating according to a message level priority scheme, the priority inversion problem associated with allowing a high priority interface to execute a low priority task, taking precedence over a lower priority interface executing a high priority task is eliminated.

[0027] Under a message level priority framework, the device I/O module (120) can control concurrent communication between a controlled device and a plurality of clients by placing messages in separate client queues associated with each client operating a virtual device interface, and moving messages from the client queues and placing the messages into a timeline in order of time and priority. Moreover, the ability to control concurrent communication is further supported by the front end interface (121) periodically determining the priority of messages in the timeline, and transmitting messages from the timeline according to priority. Furthermore, the front end interface (121) can support functionality for periodically determining a priority of messages in the timeline, detecting potential timing collisions exist for message transmission, and providing collision notification to the virtual device interfaces. Moreover, by adding a timeline for the transmit queue, it is possible to provide timely collision notification to the device interfaces to allow higher logic (such as, for example, a system controller application) to resolve the problem with the best compromise.

[0028] Although exemplary embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present system and method is not limited to those precise embodiments, and that various other changes and modifications can be affected therein by one skilled in the art without departing from the scope of the invention. All such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.

Claims

1. A method, comprising the steps of:

providing a plurality of virtual device interfaces (13) for a target device (11), wherein each virtual device interface enables client control of a discrete function of the target device;

providing a single shared I/O connection (15); and

controlling concurrent communication between the target device and two or more virtual device interfaces to operate on the single shared I/O connection using a device input/output (I/O) interface (12) to access the target device using a shared I/O module that arbitrates and coordinates I/O activity between the target device and the two or more virtual device interfaces;

further comprising the steps of:

assigning equal priority to virtual device interfaces executing tasks; and

prioritizing messages from the virtual device interfaces (13) based on assigned message-level priority;

further comprising the steps of:

placing messages in separate client queues associated with each client operating a virtual device interface (13); and
moving messages from the client queues and placing the messages into a timeline queue in order of time and priority;

further comprising the steps of:

periodically determining a priority of messages in the timeline queue;
detecting if potential timing collisions exist for message transmission; and
providing collision notification to the virtual device interfaces (13).

2. The method of claim 1, further comprising the step of:

transmitting messages from the timeline queue according to priority.

3. The method of claim 1, further comprising the step of:

using an I/O API (application programming interface) (14) that enables a client to specify the priority and timing of messages for that client.

4. The method of claim 1, further comprising the step of:

using an I/O API (application programming interface) (14) that enables a client to specify response messages received from the target device (11), which are to be provided to that client.

5. A program storage device readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps for controlling a target device (11), the method steps comprising:

using a plurality of virtual device interfaces (13) for the target device, wherein each virtual device interface enables client control of a discrete function of the target device;
using a single shared I/O connection (15) to the target device; and
controlling concurrent communication between the target device and two or more virtual device interfaces to operate on the single shared I/O connection using a device input/output (I/O) interface (12) to access the target device using a shared I/O module that arbitrates and coordinates I/O activity between the target device and the two or more virtual device interfaces;
wherein the instructions for controlling concurrent communication comprise instructions for:

assigning equal priority to virtual device interfaces executing tasks; and
prioritizing messages from the virtual device interfaces based on assigned message-level priority;

wherein the instructions for controlling concurrent communication comprise instructions for:

placing messages in separate client queues associated with each client operating a virtual device interface (13); and
moving messages from the client queues and placing the messages into a timeline queue in order of time and priority;

wherein the instructions for controlling concurrent communication comprise instructions for:

periodically determining a priority of messages in the timeline queue;
detecting if potential timing collisions exist for message transmission; and
providing collision notification to the virtual device interfaces.

6. The program storage device of claim 5, wherein the instructions for controlling concurrent communication comprise instructions for:

transmitting messages from the timeline queue according to priority.

7. The program storage device of claim 5, further comprising instructions for using an I/O API (application programming interface) (14) that enables a client to specify a priority or timing of messages for that client.

8. The program storage device of claim 5, further comprising instructions for using an I/O API (application programming interface) (14) that enables a client to specify response messages received from the target device (11), which are to be provided to that client.

9. A system, comprising:

a component that interfaces with a plurality of virtual devices (13) and a target device (11) to concurrently control communications over a single connection between two or more of the virtual devices and the target device;
a front end interface that interfaces with the virtual devices (13) and determines message queues for the virtual devices; and
a device input/output (I/O) module (12) having a front end interface (121) and an input/output (I/O) port interface (122) that interacts with a target device based on the queued messages, wherein the I/O module arbitrates and coordi-

nates I/O activity between the target device and the two or more virtual device interfaces;
characterized in that the front end interface (121) assigns equal priority to virtual devices (13) executing tasks, and prioritizes messages from the virtual devices based on assigned message-level priority;
wherein the front end interface (121) places messages in separate client queues associated with each client operating a virtual device interface, moves messages from the client queues and places the messages into a timeline queue in order of time and priority;
wherein the front end interface (121) periodically determines a priority of messages in the timeline queue; detects if potential timing collisions exist for message transmission; and provides collision notification to the virtual device interfaces.

10. The system of claim 9, wherein the front end interface (121) periodically determines a priority of messages in the timeline queue and transmits messages from the timeline queue according to time and priority.

11. The system of claim 9, further comprising:

an I/O API (application programming interface) (14) that enables a client to specify a priority and timing of messages for that client.

12. The system of claim 9, further comprising:

an I/O API (application programming interface) (14) that enables a client to specify response messages received from the target device (11), which are to be provided to that client.

Patentansprüche

1. Verfahren, das die Schritte umfasst:

Bereitstellen einer Mehrzahl von Schnittstellen (13) für virtuelle Einheiten für eine Zieleinheit (11), wobei jede Schnittstelle für virtuelle Einheiten eine Client-Steuerung einer diskreten Funktion der Zieleinheit ermöglicht;
Bereitstellen einer einzelnen gemeinsam genutzten E/A-Verbindung (15); und
Steuern einer gleichzeitigen Kommunikation zwischen der Zieleinheit und zwei oder mehr Schnittstellen für virtuelle Einheiten, um unter Verwendung einer Einheiten-Eingabe-/Ausgabe-(E/A-)Schnittstelle (12) auf der einzelnen gemeinsam genutzten E/A-Verbindung zu arbeiten, um unter Verwendung eines gemeinsam genutzten E/A-Moduls auf die Zieleinheit zuzugreifen, die eine E/A-Aktivität zwischen der Zieleinheit und den zwei oder mehr Schnittstellen für virtuelle Einheiten vermittelt und koordiniert;
das ferner die Schritte umfasst:

Zuweisen einer gleichen Priorität zu Schnittstellen für virtuelle Einheiten, die Aufgaben ausführen; und
Priorisieren von Nachrichten von den Schnittstellen (13) für virtuelle Einheiten auf der Grundlage einer zugewiesenen Nachrichtenebenenpriorität;

das ferner die Schritte umfasst:

Platzieren von Nachrichten in separaten Client-Warteschlangen, die jedem Client zugehörig sind, der eine Schnittstelle (13) für virtuelle Einheiten betreibt; und
Verschieben von Nachrichten aus den Client-Warteschlangen und Platzieren der Nachrichten nach Zeit und Priorität in einer Zeitabfolgewarteschlange;

das ferner die Schritte umfasst:

periodisches Bestimmen einer Priorität von Nachrichten in der Zeitabfolgewarteschlange;
Ermitteln, ob potenzielle Zeitabfolgekollisionen für eine Nachrichtenübertragung vorhanden sind; und
Bereitstellen einer Kollisionsbenachrichtigung an die Schnittstellen (13) für virtuelle Einheiten.

2. Verfahren nach Anspruch 1, das ferner den Schritt umfasst:

Übertragen von Nachrichten aus der Zeitabfolgewarteschlange nach Priorität.

3. Verfahren nach Anspruch 1, das ferner den Schritt umfasst:

Verwenden einer E/A-API (Anwendungsprogrammierschnittstelle) (14), die einem Client ermöglicht, die Priorität und zeitliche Planung von Nachrichten für diesen Client zu spezifizieren.

4. Verfahren nach Anspruch 1, das ferner den Schritt umfasst:

Verwenden einer E/A-API (Anwendungsprogrammierschnittstelle) (14), die einem Client ermöglicht, Antwortnachrichten zu spezifizieren, die von der Zieleinheit (11) empfangen werden, die an den Client bereitzustellen sind.

5. Programmspeichereinheit, die von einem Computer lesbar ist, die ein Programm von Anweisungen konkret verkörpert, die vom Computer ausführbar sind, um Verfahrensschritte zum Steuern einer Zieleinheit (11) durchzuführen, wobei die Verfahrensschritte umfassen:

Verwenden einer Mehrzahl von Schnittstellen (13) für virtuelle Einheiten für die Zieleinheit, wobei jede Schnittstelle für virtuelle Einheiten einem Client eine Steuerung einer diskreten Funktion der Zieleinheit ermöglicht;
Verwenden einer einzelnen gemeinsam genutzten E/A-Verbindung (15) zur Zieleinheit; und
Steuern einer gleichzeitigen Kommunikation zwischen der Zieleinheit und zwei oder mehr Schnittstellen für virtuelle Einheiten, um unter Verwendung einer Einheiten-Eingabe-/Ausgabe-(E/A-)Schnittstelle (12) auf der einen einzelnen gemeinsam genutzten E/A-Verbindung zu arbeiten, um unter Verwendung eines gemeinsam genutzten E/A-Moduls auf die Zieleinheit zuzugreifen, die eine E/A-Aktivität zwischen der Zieleinheit und den zwei oder mehr Schnittstellen für virtuelle Einheiten vermittelt und koordiniert;
wobei die Anweisungen zum Steuern einer gleichzeitigen Kommunikation Anweisungen umfasst zum:

Zuweisen einer gleichen Priorität zu Schnittstellen für virtuelle Einheiten, die Aufgaben ausführen; und
Priorisieren von Nachrichten von den Schnittstellen für virtuelle Einheiten auf der Grundlage einer zugewiesenen Nachrichtenebenenpriorität;

wobei die Anweisungen zum Steuern einer gleichzeitigen Kommunikation Anweisungen umfassen zum:

Platzieren von Nachrichten in separaten Client-Warteschlangen, die jedem Client zugehörig sind, der eine Schnittstelle (13) für virtuelle Einheiten betreibt; und
Verschieben von Nachrichten aus den Client-Warteschlangen und Platzieren der Nachrichten nach Zeit und Priorität in einer Zeitabfolgewarteschlange;

wobei die Anweisungen zum Steuern einer gleichzeitigen Kommunikation Anweisungen umfassen zum:

periodischen Bestimmen einer Priorität von Nachrichten in der Zeitabfolgewarteschlange;
Ermitteln, ob potenzielle Zeitabfolgekollisionen für eine Nachrichtenübertragung vorhanden sind; und
Bereitstellen einer Kollisionsbenachrichtigung an die Schnittstellen für virtuelle Einheiten.

6. Programmspeichereinheit nach Anspruch 5, wobei die Anweisungen zum Steuern einer gleichzeitigen Kommunikation Anweisungen umfassen zum:

Übertragen von Nachrichten aus der Zeitabfolgewarteschlange nach Priorität.

7. Programmspeichereinheit nach Anspruch 5, die ferner das Verwenden einer E/A-API (Anwendungsprogrammierschnittstelle) (14) umfasst, die einem Client ermöglicht, eine Priorität oder zeitliche Planung von Nachrichten für diesen Client zu spezifizieren.

8. Programmspeichereinheit nach Anspruch 5, die ferner Anweisungen zum Verwenden einer E/A-API (Anwendungsprogrammierschnittstelle) (14) umfasst, die einem Client ermöglichen, Antwortnachrichten zu spezifizieren, die von der Zieleinheit (11) empfangen werden, die an diesem Client bereitzustellen sind.

9. System, das umfasst:

eine Komponente, die sich mit einer Mehrzahl von virtuellen Einheiten (13) und einer Zieleinheit (11) verbindet, um Kommunikationen über eine einzelne Verbindung zwischen zwei oder mehr der virtuellen Einheiten und der Zieleinheit gleichzeitig zu steuern;
eine Front-End-Schnittstelle, die sich mit den virtuellen Einheiten (13) verbindet und Nachrichtenwarteschlangen für die virtuellen Einheiten bestimmt; und
ein Einheiten-Eingabe-/Ausgabe-(E/A-)Modul (12) mit einer Front-End-Schnittstelle (121) und einer Eingabe-/Ausgabe-(E/A-)Anschlussschnittstelle (122), die mit einer Zieleinheit auf der Grundlage der in Warteschlange gereihten Nachrichten interagiert, wobei das E/A-Modul eine E/A-Aktivität zwischen der Zieleinheit und den zwei oder mehreren Schnittstellen für virtuelle Einheiten vermittelt und koordiniert;
dadurch gekennzeichnet, dass die Front-End-Schnittstelle (121) virtuellen Einheiten (13), die Aufgaben ausführen, gleiche Priorität zuweist und Nachrichten von den virtuellen Einheiten auf der Grundlage einer zugewiesenen Nachrichtenebenenpriorität priorisiert;
wobei die Front-End-Schnittstelle (121) Nachrichten in separaten Client-Warteschlangen platziert, die jedem Client zugehörig sind, der

eine Schnittstelle für virtuelle Einheiten betreibt, Nachrichten aus den Client-Warteschlangen verschiebt und die Nachrichten in einer Zeitabfolgewarteschlange nach Zeit und Priorität platziert;
wobei die Front-End-Schnittstelle (121) periodisch eine Priorität von Nachrichten in der Zeitabfolgewarteschlange bestimmt; ermittelt, ob potentielle Zeitabfolgekollisionen für eine Nachrichtenübertragung vorliegen; und eine Kollisionsbenachrichtigung an die Schnittstellen für virtuelle Einheiten bereitstellt.

10. System nach Anspruch 9, wobei die Front-End-Schnittstelle (121) periodisch eine Priorität von Nachrichten in der Zeitabfolgewarteschlange bestimmt und Nachrichten aus der Zeitabfolgewarteschlange nach Zeit und Priorität überträgt.

11. System nach Anspruch 9, das ferner umfasst:

eine E/A-API (Anwendungsprogrammierschnittstelle) (14), die einem Client ermöglicht, eine Priorität und zeitliche Planung von Nachrichten für diesen Client zu spezifizieren.

12. System nach Anspruch 9, das ferner umfasst:

eine E/A-API (Anwendungsprogrammierschnittstelle) (14), die einem Client ermöglicht, Antwortnachrichten zu spezifizieren, die von der Zieleinheit (11) empfangen werden, die an diesen Client bereitzustellen sind.

Revendications

1. Procédé, comprenant les étapes consistant à :

fournir une pluralité d'interfaces de dispositif virtuel (13) pour un dispositif cible (11), chaque interface de dispositif virtuel permettant une commande de client d'une fonction discrète du dispositif cible ;
fournir une connexion d'entrée/sortie (E/S) partagée unique (15) ; et
commander une communication simultanée entre le dispositif cible et au moins deux interfaces de dispositif virtuel pour qu'elle se produise sur la connexion E/S partagée unique à l'aide d'une interface d'entrée/sortie (E/S) de dispositif (12) pour accéder au dispositif cible à l'aide d'un module E/S partagé qui arbitre et coordonne une activité E/S entre le dispositif cible et les au moins deux interfaces de dispositif virtuel ;
comprenant en outre les étapes consistant à :

affecter une priorité égale à des interfaces de dispositif virtuel exécutant des tâches ; et
prioriser des messages provenant des interfaces de dispositif virtuel (13) sur la base d'une priorité de niveau de message affectée ;

comprenant en outre les étapes consistant à :

placer des messages dans des files d'attente de client séparées associées à chaque client exploitant une interface de dispositif virtuel (13) ; et
déplacer des messages des files d'attente de client et placer les messages dans une file d'attente de ligne de temps par ordre de temps et de priorité ;

comprenant en outre les étapes consistant à :

déterminer périodiquement une priorité de messages dans la file d'attente de ligne de temps ;
détecter si des collisions de temporisation potentielles existent pour une transmission de message ; et
fournir une notification de collision aux interfaces de dispositif virtuel (13).

2. Procédé selon la revendication 1, comprenant en outre l'étape consistant à :

transmettre des messages de la file d'attente de ligne de temps selon une priorité.

3. Procédé selon la revendication 1, comprenant en outre l'étape consistant à :

utiliser une interface de programmation d'applications (API) E/S (14) qui permet à un client de spécifier la priorité et la temporisation de messages pour ce client.

4. Procédé selon la revendication 1, comprenant en outre l'étape consistant à :

utiliser une interface de programmation d'applications (API) E/S (14) qui permet à un client de spécifier des messages de réponse reçus à partir du dispositif cible (11), qui doivent être fournis à ce client.

5. Dispositif de stockage de programme lisible par un ordinateur, incorporant de manière tangible un programme d'instructions pouvant être exécuté par l'ordinateur afin de réaliser des étapes de procédé pour commander un dispositif cible (11), les étapes de procédé comprenant :

utiliser une pluralité d'interfaces de dispositif virtuel (13) pour le dispositif cible, chaque interface de dispositif virtuel permettant une commande de client d'une fonction discrète du dispositif cible ;
utiliser une connexion E/S partagée unique (15) vers le dispositif cible ; et
commander une communication simultanée entre le dispositif cible et au moins deux interfaces de dispositif virtuel pour qu'elle se produise sur la connexion E/S partagée unique à l'aide d'une interface d'entrée/sortie (E/S) de dispositif (12) pour accéder au dispositif cible à l'aide d'un module E/S partagé qui arbitre et coordonne une activité E/S entre le dispositif cible et les au moins deux interfaces de dispositif virtuel ;
les instructions pour commander une communication simultanée comprenant des instructions pour :

affecter une priorité égale à des interfaces de dispositif virtuel exécutant des tâches ; et
prioriser des messages provenant des interfaces de dispositif virtuel sur la base d'une priorité de niveau de message affectée ;

les instructions pour commander une communication simultanée comprenant des instructions pour :

placer des messages dans des files d'attente de client séparées associées à chaque client exploitant une interface de dispositif virtuel (13) ; et
déplacer des messages des files d'attente de client et placer les messages dans une file d'attente de ligne de temps par ordre de temps et de priorité ;

les instructions pour commander une communication simultanée comprenant des instructions pour :

déterminer périodiquement une priorité de messages dans la file d'attente de ligne de temps ;
détecter si des collisions de temporisation potentielles existent pour une transmission de message ; et
fournir une notification de collision aux interfaces de dispositif virtuel.

6. Dispositif de stockage de programme selon la revendication 5, dans lequel les instructions pour commander une communication simultanée comprennent des instructions pour :

transmettre des messages de la file d'attente de ligne de temps selon une priorité.

7. Dispositif de stockage de programme selon la revendication 5, comprenant en outre des instructions pour utiliser une interface de programmation d'applications (API) E/S (14) qui permet à un client de spécifier une priorité ou une temporisation de messages pour ce client.

8. Dispositif de stockage de programme selon la revendication 5, comprenant en outre des instructions pour utiliser une interface de programmation d'applications (API) E/S (14) qui permet à un client de spécifier des messages de réponse reçus à partir du dispositif cible (11), qui doivent être fournis à ce client.

9. Système, comprenant :

un composant qui s'interface avec une pluralité de dispositifs virtuels (13) et un dispositif cible (11) pour commander de manière simultanée des communications sur une connexion unique entre au moins deux des dispositifs virtuels et le dispositif cible ;
une interface frontale qui s'interface avec les dispositifs virtuels (13) et détermine des files d'attente de messages pour les dispositifs virtuels ; et
un module (12) d'entrée/sortie (E/S) de dispositif ayant une interface frontale (121) et une interface de port d'entrée/sortie (E/S) (122) qui interagit avec un dispositif cible sur la base des messages mis en file d'attente, le module E/S arbitrant et coordonnant une activité E/S entre le dispositif cible et les au moins deux interfaces de dispositif virtuel ;
caractérisé en ce que l'interface frontale (121) affecte une priorité égale à des dispositifs virtuels (13) exécutant des tâches, et priorise des messages provenant des dispositifs virtuels sur la base d'une priorité de niveau de message affectée ;
l'interface frontale (121) plaçant des messages dans des files d'attente de client séparées associées à chaque client exploitant une interface de dispositif virtuel, déplaçant des messages des files d'attente de client et plaçant les messages dans une file d'attente de ligne de temps par ordre de temps et de priorité ;
l'interface frontale (121) déterminant périodiquement une priorité de messages dans la file d'attente de ligne de temps ; détectant si des collisions de temporisation potentielles existent pour une transmission de message ; et fournissant une notification de collision aux interfaces de dispositif virtuel.

10. Système selon la revendication 9, dans lequel l'interface frontale (121) détermine périodiquement une priorité de messages dans la file d'attente de ligne de temps et transmet des messages de la file d'attente de ligne de temps selon le temps et la priorité.

11. Système selon la revendication 9, comprenant en outre :

une interface de programmation d'applications (API) E/S (14) qui permet à un client de spécifier une priorité et une temporisation de messages pour ce client.

12. Système selon la revendication 9, comprenant en outre :

une interface de programmation d'applications (API) E/S (14) qui permet à un client de spécifier des messages de réponse reçus à partir du dispositif cible (11), qui doivent être fournis à ce client.

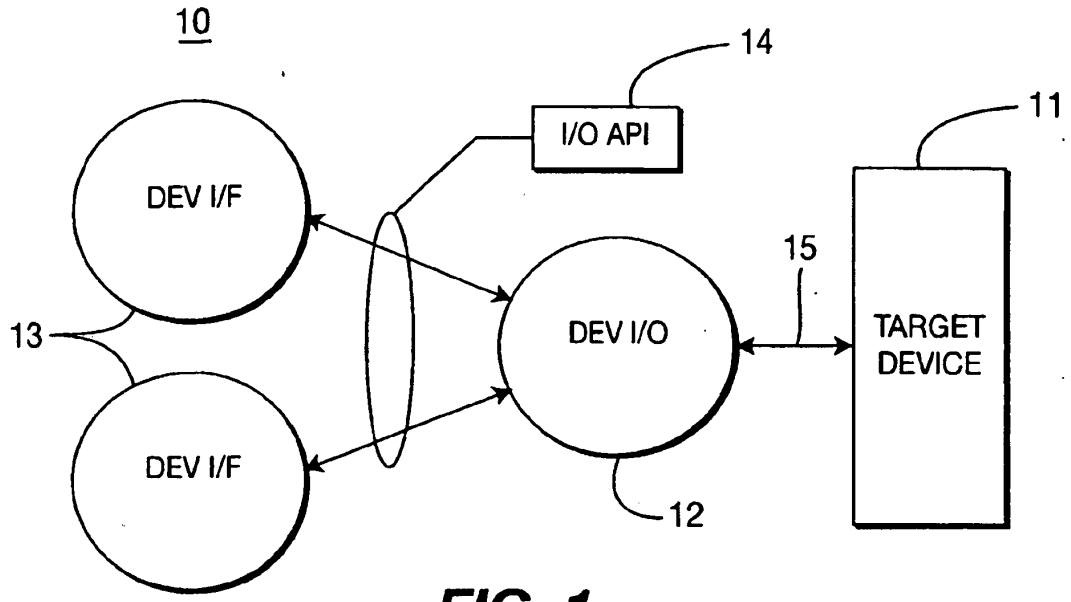

10
14
I/O API
11
DEV I/F
15
13
DEV I/O
TARGET DEVICE
DEV I/F
12

FIG. 1

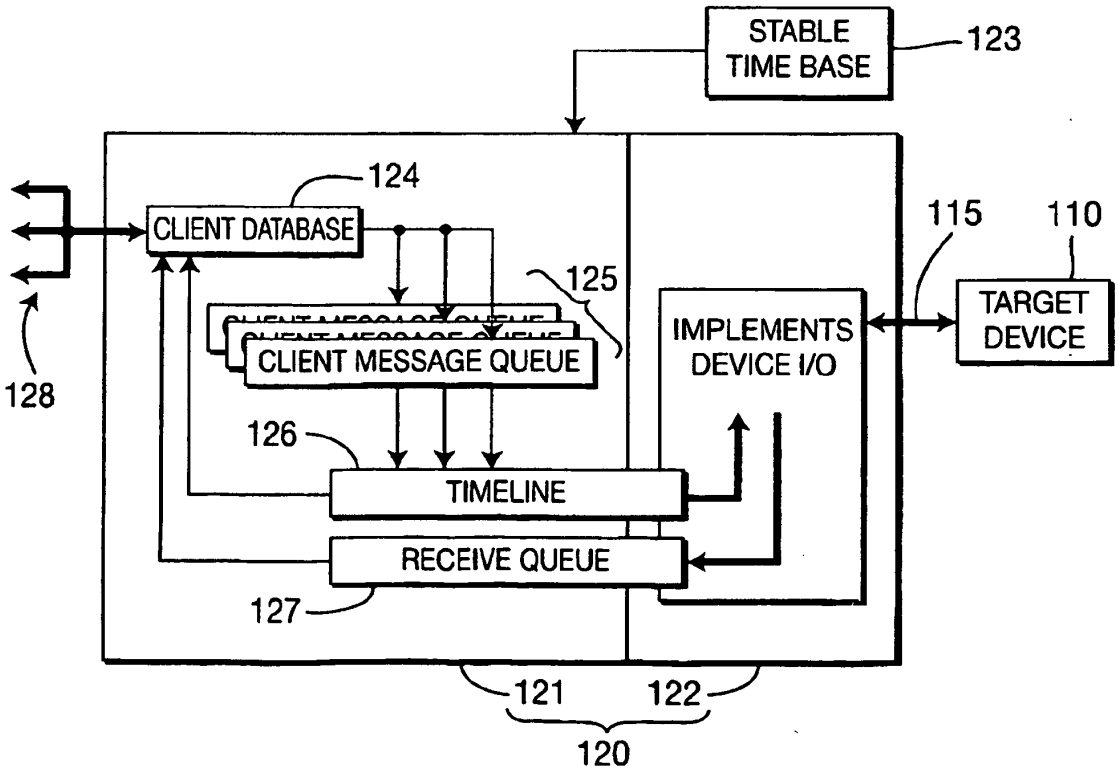

STABLE TIME BASE
123
124
CLIENT DATABASE
115
110
125
CLIENT MESSAGE QUEUE
IMPLEMENTS DEVICE I/O
TARGET DEVICE
128
126
TIMELINE
RECEIVE QUEUE
127
121
122
120

FIG. 2

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- EP 1513057 A **[0003]**
- US 20060248526 A, Rostampour **[0003]**